Connecting Distributed Version Control Systems Communities to Linked Open Data

Khaled Aslan, Hala Skaf-Molli, Pascal Molli

Connecting Distributed Version Control Systems Communities to Linked Open Data

Khaled Aslan
LINA, Nantes University, France
khaled.aslan-almoubayed@univ-nantes.fr

Hala Skaf-Molli
LINA, Nantes University, France
hala.skaf@univ-nantes.fr

Pascal Molli
LINA, Nantes University, France
pascal.molli@univ-nantes.fr

***Abstract*—Distributed Version Control Systems (DVCS) such as Git or Mercurial allow community of developers to coordinate and maintain well known software such as Linux operating system or Firefox web browser. The Push-Pull-Clone (PPC) collaboration model used in DVCS generates PPC social network where DVCS repositories are linked by *push/pull* relations. Unfortunately, DVCS tools poorly interoperate and are not navigable. The first issue prevents the development of generic tools and the second one prevents network analysis. In this paper, we propose to reuse semantic web technologies to transform any DVCS system into a social semantic web one. To achieve this objective, we propose *SCHO+* a lightweight ontology that allows to represent causal history sharing. This ontology allows each node of the PPC social network to publish semantic datasets. Next, these semantic datasets can be queried with link transversal based query execution for metrics computation and PPC social network discovery. We experimented PPC network discovery and divergence metrics on real data from some representative projects managed by different DVCS tools.**

I. Introduction

Distributed Version Control Systems (DVCS) [1] such as *git*, *Mercurial*, *Bazaar* and *Darcs* are social tools largely used in open source software development. They allow huge community of developers to coordinate and maintain softwares such as the Linux kernel project[1] and Mozilla Firefox[2].

Main characteristics of DVCS compared to traditional Version Control Systems (VCS) are decentralization and autonomous participants. Every new developer involved in a software project can set up her own code repository by cloning an existing one and start contributing by advertising new updates. This model of collaboration is called push/pull/clone (PPC) or fork/pull model. The PPC collaboration model generates networks of DVCS repositories linked by push/pull relations. These networks are very similar to social networks where a user can follow messages of others users. The main difference comes from the nature of exchanged messages i.e. in a DVCS, messages contain operations that can be applied on local files or directories. We believe that PPC social networks are difficult to observe for the following reasons:

- *DVCS tools poorly interoperate*: Although existing DVCS rely on the same basic concepts and collaboration model, they suffer from interoperability problems. Plug-ins and extra tools developed for one DVCS cannot directly be applied to another one. For instance, it is not possible to develop a metric that can be used transparently for *Git*, *Mercurial* and *Bazaar*. Some projects are also relying on combination of several VCS/DVCS. Moreover, some projects use *git* and *subversion*, other have bridges between *git* and *Bazaar*. In this case, building observation tools is even more challenging.
- *PPC social networks are not navigable*: There is no standard way for a repository to advertise their push/pull relations. Actually the push/pull relations are private and not even published to others. This prevent the discovery of the DVCS social network. This is not a requirement for DVCS i.e. every user is free to pull changes from any source she wants, she is also free to keep this information private.

These two issues have important consequences for software developers involved in software projects. For example: i) clustering can occur within the PPC social network if one developer shuts down her repository; ii) estimating the global activity of the PPC social network is also important for project management, awareness and coordination.

Navigability of PPC social network can be acheived if all DVCS repositories for a software project are hosted within a single software forge such as *GitHub*, *launchpad*, *sourceforge* [3]. Github allows navigation between Git repositories hosted on Github. Of course, this approach is very restrictive i.e. navigability of PPC social networks should not rely on forge providers.

Another approach is to reuse semantic web technologies and transform any DVCS into a social semantic web tool. Semantic web technologies are inherently distributed and offer strong support for interoperability.

In order to overcome problems of DVCS interoperability, we propose a lightweight ontology *SCHO+*. *SCHO+* is based on the observation that existing DVCS follow the optimistic replication model [9]. *SCHO+* conceptualizes causal histories and push/pull relations.

Based on *SCHO+*, we give the opportunity to PPC actors to extract informations from their local DVCS repositories and generate RDF datasets. These data are clearly targeted for the

[1]http://www.kernel.org/
[2]http://www.mozilla.com/
[3]https://github.com https://launchpad.net https://sf.net

linked data and reuse FOAF[4]/DOAP[5] vocabularies. Next, each PPC actor can perform queries on the PPC social network using Link Traversal Based Query Execution [6].

In order to validate our approach, we populate the *SCHO+* ontology with real causal histories from *git*, *Mercurial* and *Bazaar* repositories and compute the same divergence metrics on the different datasets. In order to validate PPC social network requests, we experiment the discovery of the PPC network for a group of developers.

The paper is structured as follows. Section II presents related work. Section III gives an overview of the proposed approach. Section IV details the main concepts and properties of *SCHO+* ontology. Section V presents the validation of the contribution by using real data from different DVCS. The last section concludes the paper.

II. Related Work

Distributed Version Control Systems (DVCS) [1] are social tools largely used in open source software development. They focus on sharing changes between autonomous participants by using push/pull/clone (PPC) or fork/pull model. DVCS tools suffer from interoperability problems and the PPC social networks are not navigable.

To overcome the interoperability problems, existing solutions are focusing on the definition of standards. For instance, the Ontology Definition Meta Model (ODM) [6] is a standard of Object Management Group (OMG) to integrate ontology language into the software development process, the Open Services for Life-cycle Collaboration (OSLC) community [7] proposes standards to define the way that life-cycle tools can share data (for example, requirements, defects, test cases, plans, or code) with one another. The objective is to make it easier for development teams to use life-cycle tools in combination and more efficiently share information between systems. For instance, OSLC Software Configuration Management (SCM) defines a common set of resources, formats and RESTful services to access and manipulate software configuration management. The scope of OSLC SCM is larger than the scope of *SCHO+* approach. The *SCHO+* considers a higher level of abstraction of DVCS by considering only one facet of these tools. The *SCHO+* approach allows to externalize the causal histories as RDF datasets. This is compatible with the OSLC SCM visions and can be integrated in OSLC SCM proposal. The main differences between *SCHO+* and OSLC are:

- OSLC is state based, while *SCHO+* is operation based which makes it independent of the data model.
- OSLC does not define social relations between the users.
- OSLC is dedicated to software engineering software while *SCHO+* is a general framework for any multi-synchronous collaboration system.

[4]http://www.foaf-project.org/
[5]http://trac.usefulinc.com/doap
[6]http://www.omg.org/spec/ODM/
[7]http://open-services.net/

To make the PCC social networks navigable, existing approaches rely on providers that host all the PPC social networks. Software forges such as *GitHub*, *launchpad*, *sourceforge* can play partially this role. For instance, the interactive *GitHub* network graph visualizer provides a to-do list of the code for registered users involved in a collaborative software project. This graph represents only a small part of the PPC social network. In addition, *GitHub* is based on a centralized architecture and closed code. The *SCHO+* approach allows to distribute PPC nodes on different providers and maintain the PPC social network navigability. This is more compatible with the distributed and autonomous nature of open source development projects.

In previous work [2], we defined the shared causal history ontology *SCHO* that manages the causal history sharing. *SCHO* ontology defines all the basic concepts common to DVCS: *ChangeSet, Patch, Previous, Operation,* etc. It also defines concepts that allow to support the PPC model: *PullFeed* and *PushFeed*. In [2], the proposed ontology was used to demonstrate how it is possible to compute all divergence awareness metrics proposed by the Computer Supported Cooperative Work (CSCW) community for *Git* repositories. *SCHO+* is an extension of *SCHO* that allows to link participants and objects by using FOAF/DOAP vocabularies and links the DVCS community to the LOD cloud. In this work, we go further we show how it is possible to develop a tool that can compute all existing divergence awareness metrics proposed by the CSCW community for *Git, Mercurial and Bazaar* and we propose a tool to analyze the PCC social network.

III. Background and Motivations

A. The Push/Pull/Clone Model

Users of DVCS interact thanks to the *Push/Pull/Clone* (PPC) model. Basically, the *Clone* operation allows users to create a local repository of an existing repository, therefore user can work isolated on her local repository. The *Push* operation allows to make public the local modifications, and finally the *Pull* operation allows to integrate remote modifications. The cycle of *Push/Pull* and therefore divergence/convergence will repeat throughout the project life.

Concretely, when developers use a DVCS software they can work as the scenario depicted in figure 1. In this scenario, two developers bob and $alice$ working on two different sites: $Site_1$ and $Site_2$.

1) bob initializes his private repository.
2) bob clones his private repository into a public one so he can publish his local modifications on it.
3) bob modifies his private repository locally.
4) bob publishes his modification into his public repository.
5) $alice$ wants to collaborate with bob on the same project. She clones bob public repository into her own private repository.
6) $alice$ modifies her private repository locally.
7) $alice$ creates a public repository by cloning her private repository to publish her modifications.

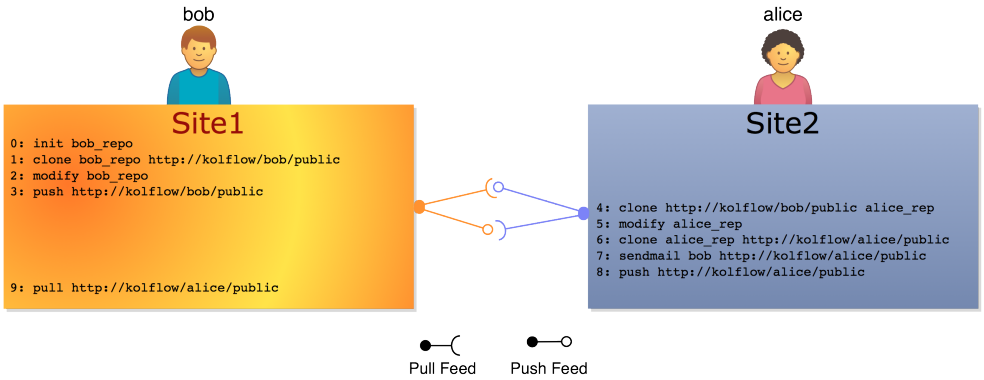


Fig. 1: Push/Pull/Clone Model

8) *alice* communicates her public repository URL to *bob* (by email for example).
9) *alice* pushes the modifications done on her private repository to her public repository.
10) *bob* pulls the modifications done on the public repository of *alice* into his private repository. This allows to maintain the two repositories synchronized and reduces the divergence.

Although existing DVCS rely on the PPC model and have the same basic concepts, they suffer from interoperability problems. Working using the PPC model creates a network of collaborators, but there is no standard way for the DVCS to publish the push/pull relationships. Nowadays, this network is hidden and we can not run any query on it. Discovering the collaboration relations is important to push further the collaboration between people. It is also important to evaluate the location of actors in the network [4]. This will give us insight on the collaboration activity which is crucial for project management. To leverage this problem we will use semantic web technologies that will provide a lightweight common ontology and render this network queryable. This can be achieved because all DVCS rely on the optimistic replication model so we can have an abstraction based on this model. We will detail this model in the following section.

B. DVCS and Optimistic Replication

DVCS follow the optimistic replication model [9]. This model considers sites where any kind of objects are replicated. Objects can be modified anytime, anywhere by applying an update operation locally. Every operation follows the following lifecycle:

1) An operation is generated in one site. It is executed immediately without any locking scheme, even if the local site is off-line.
2) It is broadcasted to all other sites. The broadcast is supposed reliable. All generated operations eventually arrive to all sites.
3) Received operations are integrated and re-executed.

The correctness of the system is defined by properties such as causality, eventual consistency, intention preservation [10]. All DVCS ensure at least causal consistency [7]. Causal consistency ensures that if a site has executed an operation op_1 before another operation op_2, then all sites will execute these two operations in the same order. More formally, we define the relation "happened-before"

Definition 1 (happened-before): Given two operations op_1, op_2 generated respectively on site i and j. $op_1 \rightarrow op_2 \Leftrightarrow$ 1- $i = j$ and op_1 has been generated before the generation of op_2, or 2- $i \neq j$ and op_1 is the sending of a message by one process, and op_2 is the receipt of that message by another process, or 3- $\exists op_3 : (op_1 \rightarrow op_3) \wedge (op_3 \rightarrow op_2)$

A system ensures causal consistency if all sites execute the same set of operation ensuring the "happened before" relation. The traditional way to implement causal consistency is to implement a causal reception i.e. an operation is delivered to the local processes if all causal operations have been delivered before. In distributed system, causal reception is implemented traditionally with vector clocks [8].

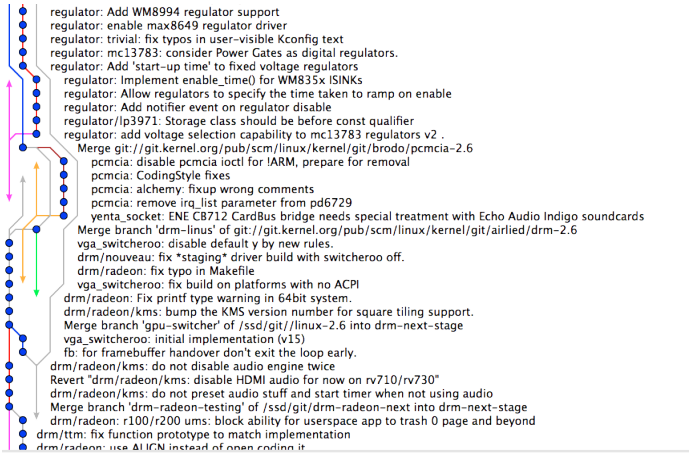


Fig. 2: Causal history in Git

DVCS follow the optimistic replication model:

1) Each repository is a site where objects i.e. files and directories are replicated.
2) Object can be modified anytime, anywhere by applying

operations. In DVCS, this is achieved by generating "commit objects" that can be interpreted as a set of operations updating several objects.

3) Operations are broadcasted to other sites. In DVCS, broadcast is achieved through push/pull operations. This can be interpreted as an anti-entropy mechanism that is part of epidemic protocols [3]. Anti-entropy protocols ensure causal delivery of operations.
4) Causal relationships are not represented with vector clocks that require join and leave procedure, but with explicit "previous relationships" between "commit objects". These relations can be observed in the graph of figure 2. This figure presents a fragment of the causal history of the linux kernel.
5) DVCS ensure at least causal consistency i.e. all repositories execute the same operations respecting the same causal order. As causal order is a partial order, it does not mean that all site has the same execution history, but they will all converge to the same causal graph.

The proposed *SCHO+* ontology conceptualize the "previous relation" and push/pull relations and consequently take an abstraction level that is able to unify existing DVCS. We detail this aspect in section IV.

C. A Motivating Example

Generating RDF [8] triples and linking the RDF graphs of the different sites will enable us to query and discover the PPC social network as shown in figure 3. In this scenario, four sites are using DVCS and are connected to each other with push/pull links. These links can be seen as an ad-hoc P2P network. In this scenario :

- $Site_1$ and $Site_2$ are engaged in a push/pull from each other. This means that $Site_1$ is pushing its modifications to $Site_2$ and $Site_2$ is accepting these modifications, and vice versa.
- $Site_3$ pushes modifications to $Site_4$ and pulls from $Site_1$.
- $Site_4$ pushes modifications to $Site_1$ and pulls from $Site_3$.

The push/pull interactions in the DVCS generate triples based on a common ontology at each site. These triples are stored in an RDF file that have an accessible URI. This way a user on a given site can run a distributed SPARQL query to explore the PPC social network. Or she can run a divergence awareness metric query to capture the network activity. A user can link her *foaf:profile* and the project description *doap:project*. So her data will be available to the whole LOD community.

We expect each DVCS user to run a script that will publish some information about the current state of his personal workspace. This information will be published as an RDF file conform to the *SCHO+* ontology. Next each user can run semantic queries relying on Link Traversal Based Query Execution.

[8]http://www.w3.org/TR/rdf-primer/

IV. SCHO+: Extending the Shared Causal History Ontology

The Shared Causal History Ontology (SCHO) [2] is an ontology for sharing and managing causal history. SCHO ontology defines all the basic concepts common to DVCS such as ChangeSet, Patch, Previous, Operation, etc. It also defines more precise concepts such as PullFeed and PushFeed to support the PPC model. The existence of a PullFeed on *Site1* that consumes operations from a PushFeed on *Site2* can be interpreted as *follow* relation between the two sites, i.e. *Site1 follow Site2*.

SCHO+ is an extension of *SCHO* that allows to link participants and objects by using FOAF/DOAP vocabularies and links the DVCS community to the LOD cloud. We add a new class *User* presented in listing 1. We link it to the FOAF profile of changesets authors using an owl:DatatypeProperty *foafProfile*. We add a new class *Project* presented in listing 2. We link it to the DOAP description for each project using an owl:DatatypeProperty *doapDesc*. We also add a new owl:ObjectProperty *relatedPush* presented in listing 3. This owl:ObjectProperty links the *PullFeed* to its corresponding *PushFeed*. This will enable us to extract the *follow* relation between the sites that generated these feeds. We will use this *follow* relation to discover the PPC social network between the different sites. Figure 4 shows the *SCHO+* ontology.

```
<owl:Class rdf:about="#User">
        <rdfs:subClassOf rdf:resource="&owl
            ;Thing"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="#
    foafProfile">
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:range rdf:resource="&xsd;
            anyURI"/>
</owl:DatatypeProperty>
```

Listing 1: New class User

```
<owl:Class rdf:about="#Project">
        <rdfs:subClassOf rdf:resource="&owl
            ;Thing"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="#doapD">
        <rdfs:domain rdf:resource="#Project
            "/>
        <rdfs:range rdf:resource="&xsd;
            anyURI"/>
</owl:DatatypeProperty>
```

Listing 2: New class Project

```
<owl:ObjectProperty rdf:about="#relatedPush
    ">
        <rdfs:range rdf:resource="#PullFeed
            "/>
        <rdfs:domain rdf:resource="#
            PushFeed"/>
</owl:ObjectProperty>
```

Listing 3: Related Push Property

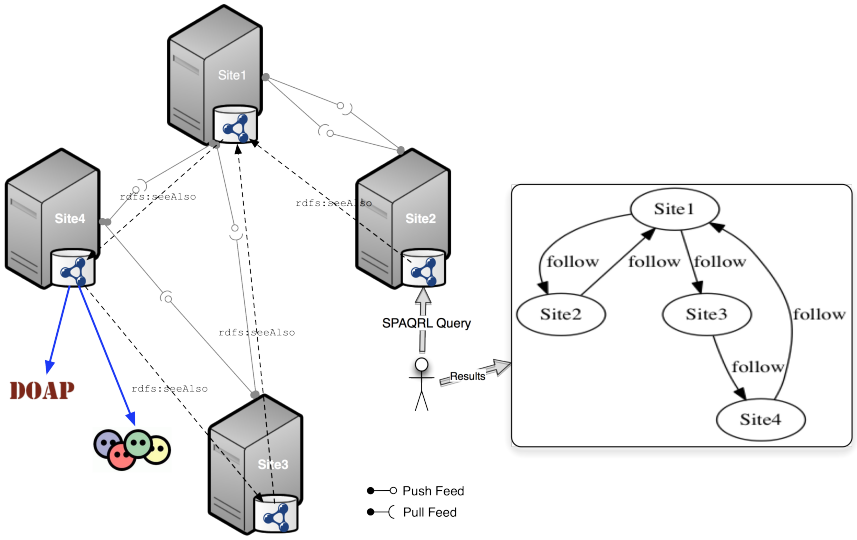


Fig. 3: Push/Pull Network

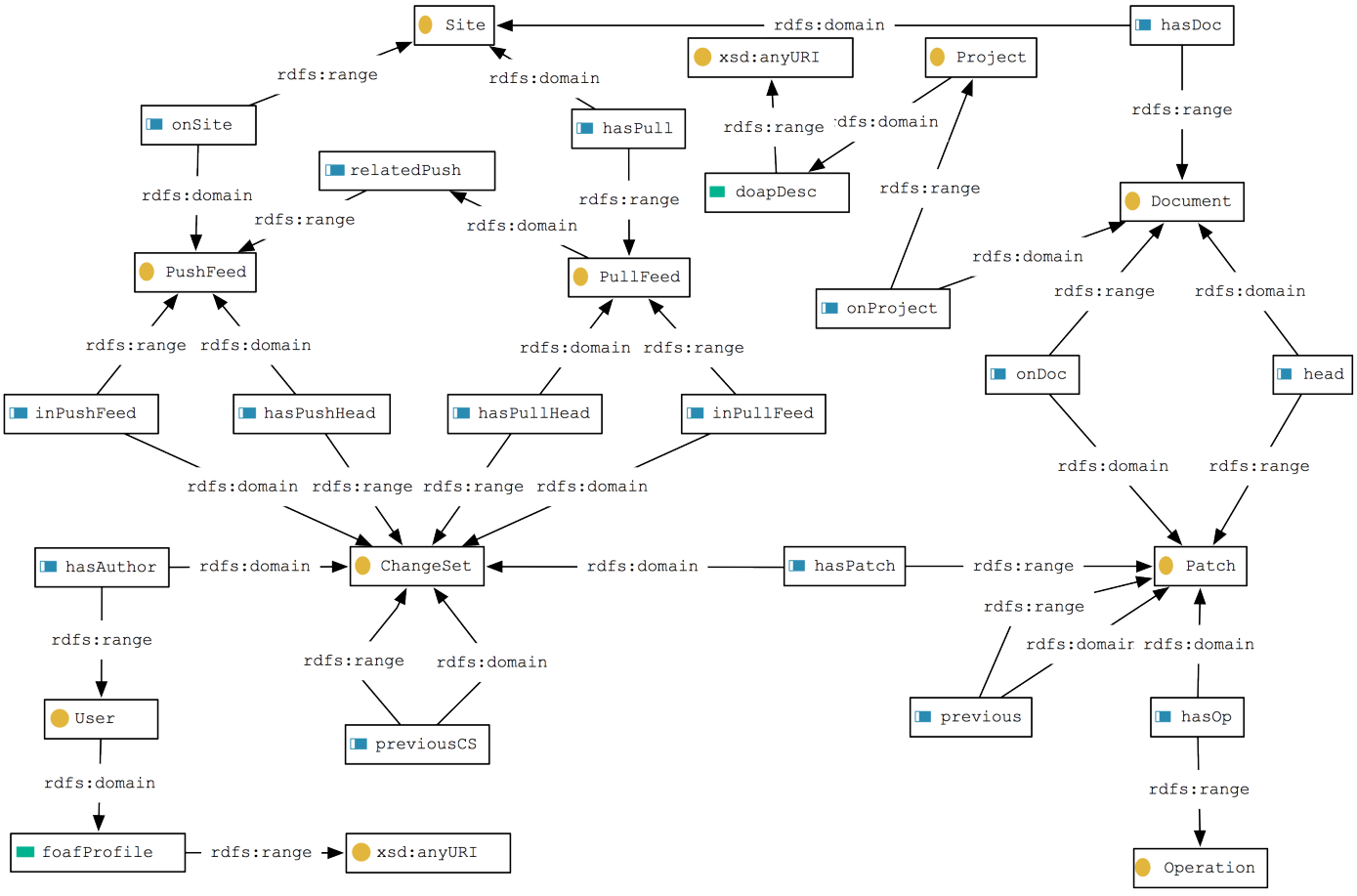


Fig. 4: Shared Causal History Ontology Extension

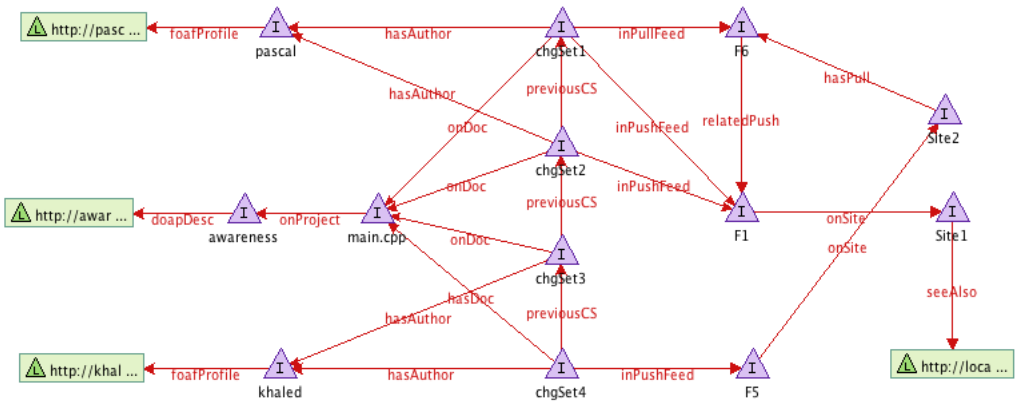


Fig. 5: $Site_2$ RDF graph

The advantages of using *SCHO+* ontology are:

- First we have a unified minimal ontology for representing and managing the shared causal history of any DVCS. This will make it easier to develop universal tools and plug-ins for any DVCS software that adopts this ontology for managing its log. So we can run queries directly on any DVCS system that uses *SCHO+* ontology without the need to import/export histories between different DVCS tools.
- Furthermore we can have generic analysis tools which can be run over this log to discover the underlying dynamics of the corresponding network. We have also linked the ontology to the LOD using the DOAP and FOAF ontologies. This will permit to link the DVCS communities with LOD and will give them a higher visibility. In order to be included in the analysis and statistics done on the LOD.

Figure 5 shows the corresponding RDF graph of $Site_2$ from the scenario presented in figure 3.

In the following section, we detail the queries that allow each user to compute divergence awareness metrics and extract the PPC social network.

V. Validation

In order to validate our approach we first demonstrate how it is possible to build a general tool that can be used for all the DVCS using the *SCHO+* ontology. Then we show how linking the RDF graphs can increase the ability to discover the PPC social network without the need for a centralized node.

We populated the *SCHO+* ontology with causal history data from different DVCS. We used *git*, *Mercurial* and *Bazaar* repositories. These repositories have rich sets of data of different size.

To use the DVCS data, first we had to inject the log data into a triple store to populate our ontology. We used the Jena TDB[9] triple store, then we implemented a parser called *dvcs2lod*[10]. *dvcs2lod* is responsible for the mapping between the concepts defined in the DVCS log and the *SCHO+* ontology. This parser can handle *git*, *Mercurial* and *Bazaar* repositories. We also added *rdfs:seeAlso* annotations to use the Link Traversal Based Query Execution [6].

A. Divergence awareness computation

The general tool that we will show is a divergence awareness tool: *DAtool*[11]. *DAtool* calculates the divergence awareness metrics using the *SCHO+* ontology.

We re-use the metrics defined in [2], which are generic divergence awareness metrics, and we apply them on different open source projects that use different DVCS. We use SPARQL [12] queries to calculate divergence awareness. For example the query shown in listing 4 returns the state *Remotely Modified* for a ChangeSet *$CSid*.

```
SELECT ?pf  WHERE {
       scho:$CSid scho:inPullFeed ?pf .
       scho:$CSid scho:date ?date .
       ?pf scho:hasPullHead ?CSHead
       scho:?CSHead scho:date ?headDate .
       NOT EXISTS { scho:$CSid scho:
           published "true".}
       FILTER ( xsd:dateTime(?headDate) <=
           xsd:dateTime(?date) )
       }
```

Listing 4: Remotely Modified ChangeSet SPARQL Query

We used real projects to validate the approach, such as: gollum [13], HgView [14], Murky [15], AllTray [16], Anewt [17] and MongoDB [18]. These projects use different DVCS such as *git*, *Mercurial* and *Bazaar*. Table I shows the details of each project and the execution time for populating the ontology with the causal history of these projects. In addition, we calculate

[9] http://openjena.org/

[10] *dvcs2lod* source code is available at: https://github.com/kmobayed/dvcs2lod

[11] *DAtool* source code is available at: https://github.com/kmobayed/DAtool

[12] http://www.w3.org/TR/sparql11-query/

[13] github.com/github/gollum.git

[14] www.logilab.org/project/hgview

[15] bitbucket.org/snej/murky/wiki/Home

[16] launchpad.net/alltray

[17] anewt.uwstopia.nl

[18] www.mongodb.org

Project name	DVCS	#CS	#Users	#Triples	Time (sec)
Gollum	Git	613	37	2851	12
MongoDB	Git	13636	91	68186	158
AllTray	Bazaar	389	3	2168	5
Anewt	Bazaar	1980	13	9433	44
hgview	Mercurial	595	15	3257	12
murky	Mercurial	198	17	1111	5

TABLE I: Execution time and general statistics

the number of ChangeSets, Users, Merges and the number of triples generated based on the *SCHO+* ontology.

Figure 6 shows the results obtained after calculating the divergence awareness metrics on the selected projects. The *Y-axis* represents the number of changesets, while the *X-axis* represents the time. In each graph, we see the number of locally modified changesets (LM) and the number of the remotely modified changesets (RM) at a given time. We can clearly observe the periods of convergence and divergence. This clearly demonstrate that the same divergence awareness metrics can be represented and computed on data produced by different DVCS.

B. Network Discovery

Linking the RDF graphs using the *SCHO+* ontology will allow the discovery of the PPC social network without the need for a centralized node or a social service provider. The social service provider has access to all the data which arises privacy and censorship issues [5], since it can exploit the whole PPC social network relations and interactions among the users. In order to overcome these issues, new decentralized approaches were proposed. They provide collaborative services without a dedicated service provider. Users can create their own collaborative network and share the collaborative services offered by the system using their own resources. If it is easy for a centralized node to extract the complete social network graph from the observed interactions. Obtaining social network informations in the distributed approach is more challenging. In fact, the distributed approach is designed to protect privacy of users and thus makes extracting the whole social network difficult.

We will show how linking the RDF graphs would make the PPC social network discovery easier. On one hand, this social network is independent of the project that the user is working on. On the other hand, this social network is also independent of the used DCVS. i.e. the we will have a higher level of abstraction for this PPC social network. We add a semantic annotation using the *rdfs:seeAlso*[19] property to each site. This annotation will include the URI of the RDF file of each site.

In fact, using the *SCHO+* ontology renders discovering the network, no more than executing a SPARQL query. We will use the Link Traversal Based Query Execution approach [6] to execute this query. The advantages of this approach are: There is no need to know all the data sources in advance. The queried data will be up-to-date. And it is independent of the existence of SPARQL endpoints provided by the data sources. Listing 5 shows this query.

```
:Site1     a scho:Site;
     scho:hasPull :F2,
                  :F4 .
:Site2     a scho:Site;
     rdfs:seeAlso <site2_RDF_URI> .
:Site3     a scho:Site;
     rdfs:seeAlso <site3_RDF_URI> .
:Site4     a scho:Site;
     rdfs:seeAlso <site4_RDF_URI> .
:F1      a scho:PushFeed;
     scho:onSite :Site1 .
:F2      a scho:PullFeed;
     scho:relatedPush :F5 .
:F3      a scho:PushFeed;
     scho:onSite :Site1 .
:F4      a scho:PullFeed;
     scho:relatedPush :F8 .
:F5      a scho:PushFeed;
     scho:onSite :Site2 .
:F8      a scho:PushFeed;
     scho:onSite :Site4 .
```

(a) $Site_1$ RDF file

```
:Site2     a scho:Site;
     scho:hasPull :F6 .
:Site1     a scho:Site;
     rdfs:seeAlso <site1_RDF_URI> .
:F5      a scho:PushFeed;
     scho:onSite :Site2 .
:F6      a scho:PullFeed;
     scho:relatedPush :F1 .
:F1      a scho:PushFeed;
     scho:onSite :Site1 .
```

(b) $Site_2$ RDF file

Fig. 7: Scenario example RDF files

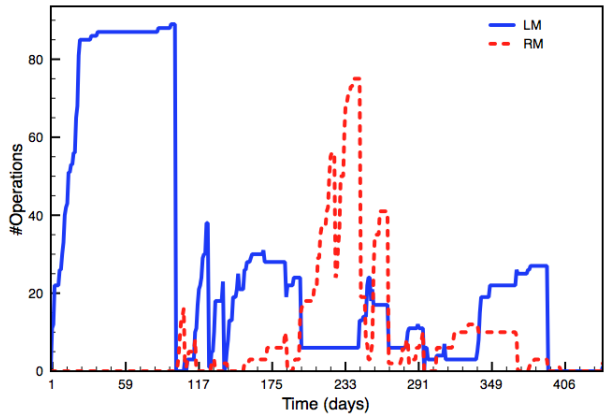
```
SELECT DISTINCT ?site1 ?site2 WHERE {
        ?site1 a scho:Site .
        ?site2 a scho:Site .
        ?pull a scho:PullFeed .
        ?push a scho:PushFeed .
        ?pull scho:relatedPush ?push .
        ?push scho:onSite ?site1 .
        ?site2 scho:hasPull ?pull .
        FILTER (?site1 != ?site2)
        }
```

Listing 5: Network Discovery SPARQL Query

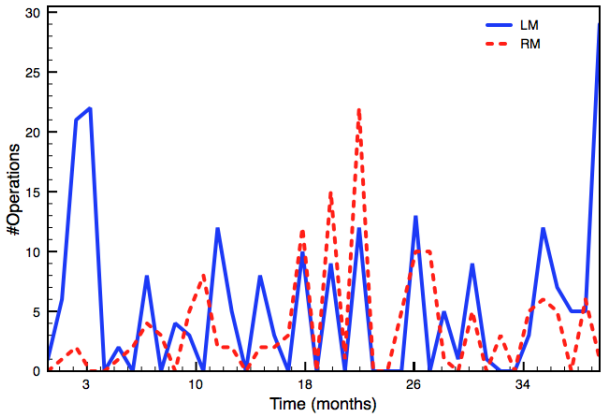
We will take the previous example presented in figure 3. In this example, we have $Site_2$ collaborates with $Site_1$ but it has no direct knowledge about the whole network. Figure 7 shows snapshots of the RDF files present on $Site_1$ and $Site_2$. We will extract the PPC social network using the SPARQL query in Listing 5. This query will give us a list of sites that have a collaboration link among them. We visualize the output using *graphviz*[20] graph visualization software. First, we run the query over $Site_2$ RDF file without the *rdfs:seeAlso* annotation,

[19]http://www.w3.org/TR/rdf-schema/

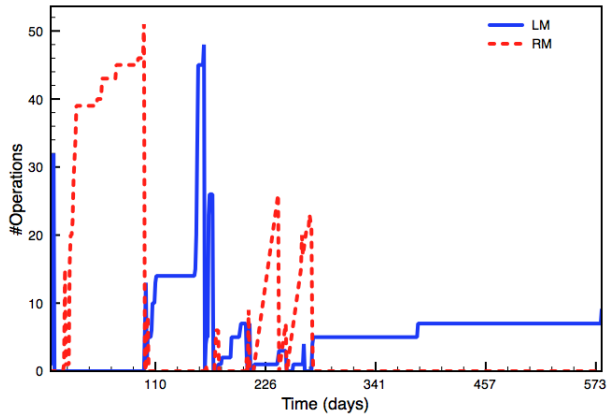
[20]http://www.graphviz.org/



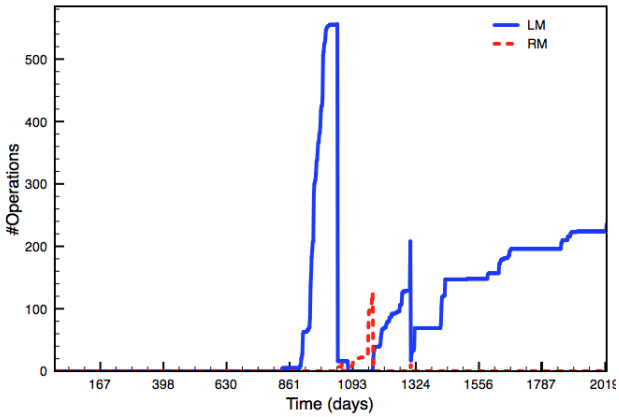
(a) gollum project (git)



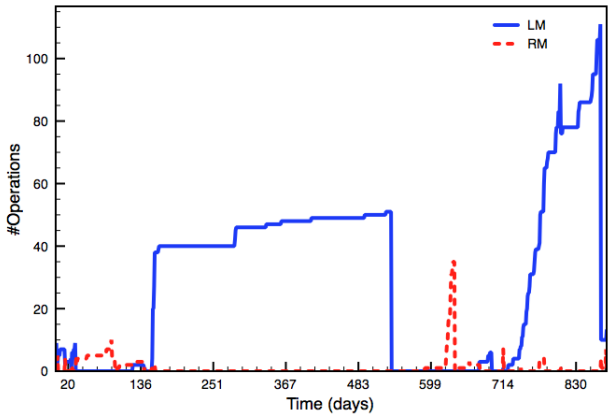
(b) mongoDB project (git)



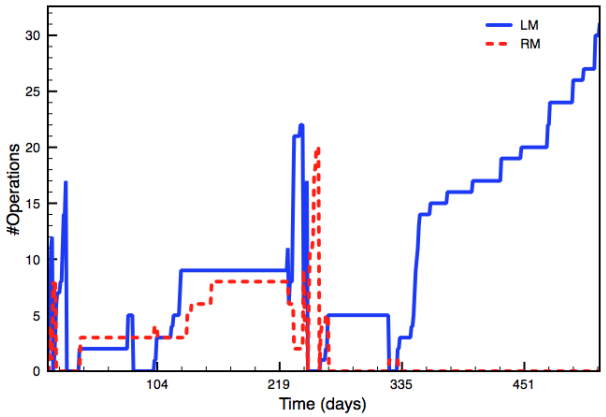
(c) AllTray project (Bazaar)



(d) Anewt project (Bazaar)

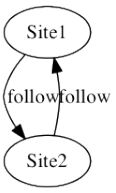


(e) hgview project (Mercurial)

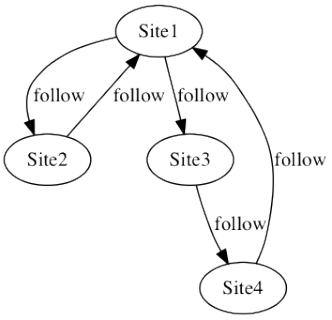


(f) Murky project (Mercurial)

Fig. 6: Divergence awareness results for different open source projects



(a) $Site_2$ PPC social network without linking the RDF files



(b) $Site_2$ PPC social network with linking the RDF files

Fig. 8: Network discovery

see figure 8a. Next, we run the same query over $Site_2$ RDF file but this time with the *rdfs:seeAlso* annotation, see figure 8b.

VI. Conclusion and Future Work

Distributed version control systems (DVCS) rely on the powerful PPC collaboration model. This model is intrinsically decentralized and builds a PPC social networks. This networks can be compared to traditional social networks where a person can follow other people. In this paper, we pointed out the problems of interoperability of DVCS tools and navigability of PPC social networks. We proposed the *SCHO+* ontology that captures one facet of DVCS. *SCHO+* is conceptualization of DVCS as an instance of optimistic replication model. *SCHO+* externalizes causal histories and push/pull relations and allows some metrics computation and PPC network discovery.

Compared to related works, in this work we do not want to solve completely the interoperability problem of DVCS tools as in OSLC approach. We just propose an ontology that captures one facet of DVCS and demonstrate that it is sufficient to compute interesting queries on heterogeneous set of DVCS tools. Compared to forge approaches such as GitHub or Launchpad, our approach allows to distribute PPC nodes on different providers and maintain the PPC social network navigability. The experimentations show that it is possible to rely on Link Traversal Query Based Execution to compute metrics and PPC social network discovery.

These results open new issues and perspectives. The first issue concerns performance and scalability. Network discovery relies on Link Traversal Based Query Execution. The performance of queries will degrade proportionally with the size of the network. Performance evaluation is needed to determine the usability threshold and proposes cache techniques for optimization. The second issue concerns privacy. Revealing pull information will imply revealing push information which is not owned by the pull owner. Currently there is no privacy policy attached to push information. Network discovery should take into account privacy policies attached to push information.

These results also open several perspectives. First, the results of divergence metrics can be analyzed to find patterns of divergence and extract some best practice. Second, it is possible to host directly *SCHO+* rdf files on DVCS hosting providers in order to demonstrate how it is possible to deploy PPC social network across different DVCS hosting providers. Finally, we demonstrate how it is possible to execute distributed queries on PPC social networks, this opens the door for distributed FLOSS Metrics computation and more generally to in depth PPC social networks analysis.

References

[1] L. Allen, G. Fernandez, K. Kane, D. Leblang, D. Minard, and J. Posner. ClearCase MultiSite: Supporting Geographically-Distributed Software Development. *Software Configuration Management: Scm-4 and Scm-5 Workshops: Selected Papers*, 1995.

[2] Khaled Aslan, Nagham Alhadad, Hala Skaf-Molli, and Pascal Molli. SCHO: An Ontology Based Model for Computing Divergence Awareness in Distributed Collaborative Systems. In *The Twelfth European Conference on Computer-Supported Cooperative Work*, Aarhus, Denmark, 2011.

[3] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, pages 1–12, New York, NY, USA, 1987. ACM.

[4] L Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1979.

[5] Ralph Gross, Alessandro Acquisti, and H. John Heinz, III. Information revelation and privacy in online social networks. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 71–80, New York, NY, USA, 2005. ACM.

[6] Olaf Hartig, Christian Bizer, and Johann Christoph Freytag. Executing sparql queries over the web of linked data. In *International Semantic Web Conference*, pages 293–309, 2009.

[7] Leslie Lamport. Times, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

[8] Friedemann Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[9] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1):42–81, March 2005.

[10] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5(1), 1998.